The `isl` Scheduler

Sven Verdoolaege

KU Leuven and Polly Labs

March 14, 2018

Outline

Outline

Overview of isl

isl is a thread-safe C library for manipulating **integer sets and relations**

- bounded by *affine constraints*
- involving *symbolic constants* and
- *existentially quantified variables*

plus **quasi-affine** and **quasi-polynomial functions** on such domains

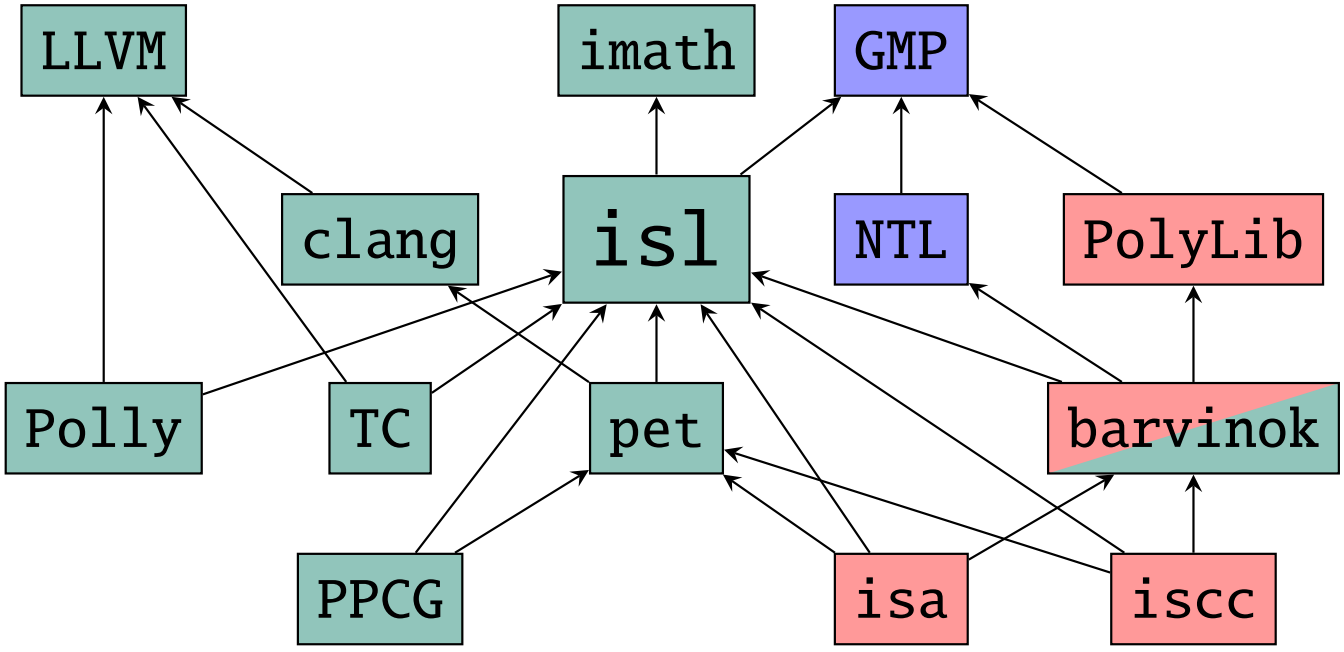
Supported operations by core library include

- *intersection*
- *union*
- *set difference*
- *integer projection*
- *coalescing*
- *closed convex hull*
- *sampling*, *scanning*
- *integer affine hull*
- *lexicographic optimization*
- *transitive closure* (approx.)
- *parametric vertex enumeration*
- *bounds on quasipolynomials*

Polyhedral compilation library

- *schedule trees*
- *dataflow analysis*
- *scheduling*
- *AST generation*

Connection with other Libraries and Tools



Licenses:
BSD/MIT/
Apache
LGPL
GPL

`isl`: manipulates parametric affine sets and relations
`barvinok`: counts elements in Presburger sets and relations
`pet`: extracts polyhedral model from clang AST
`PPCG`: Polyhedral Parallel Code Generator
`iscc`: interactive calculator
`TC`: Tensor Comprehensions

Set Representation

[12]

```
S:   A[0] = 1;
     for (i = 1; i < N; ++i)
T:       A[i] = 2 * A[i - 1];
```

Set Representation

[12]

```
S:   A[0] = 1;
     for (i = 1; i < N; ++i)
T:       A[i] = 2 * A[i - 1];
```

- isl: named (and nested) spaces

```
[N] -> { S[]; T[i]: 1 <= i < N }
```

Set Representation

[12]

```
S:   A[0] = 1;
     for (i = 1; i < N; ++i)
T:       A[i] = 2 * A[i - 1];
```

- isl: named (and nested) spaces

```
[N] -> { S[]; T[i]: 1 <= i < N }
```

- omega:

```
symbolic N;
{ [0, 0] } union { [1, i]: 1 <= i < N}
```

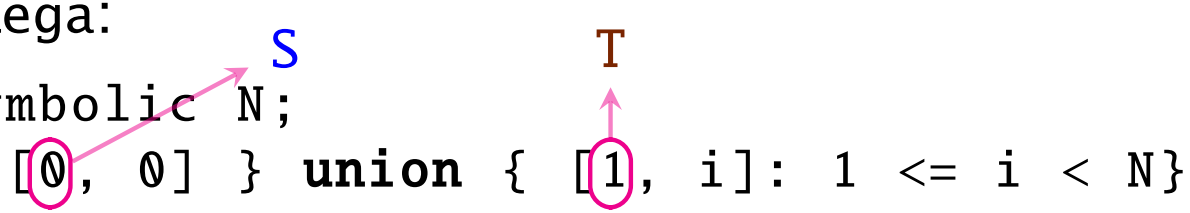
Set Representation


```
S:   A[0] = 1;
     for (i = 1; i < N; ++i)
T:       A[i] = 2 * A[i - 1];
```

- isl: named (and nested) spaces

```
[N] -> { S[]; T[i]: 1 <= i < N }
```

- omega:



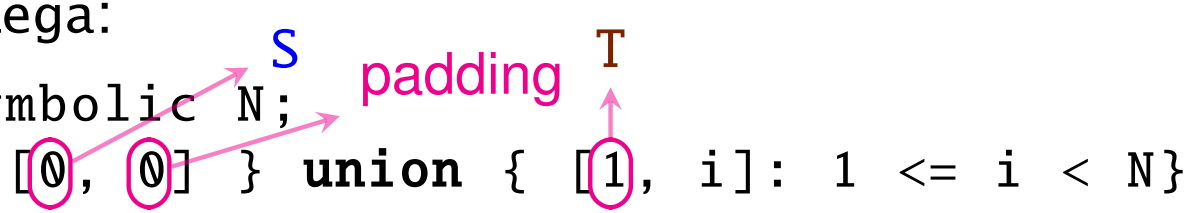
Set Representation


```
S:   A[0] = 1;
     for (i = 1; i < N; ++i)
T:       A[i] = 2 * A[i - 1];
```

- isl: named (and nested) spaces

```
[N] -> { S[]; T[i]: 1 <= i < N }
```

- omega:



Set Representation


```
S:   A[0] = 1;
     for (i = 1; i < N; ++i)
T:       A[i] = 2 * A[i - 1];
```

- isl: named (and nested) spaces

```
[N] -> { S[]; T[i]: 1 <= i < N }
```

- omega:

```
symbolic N;
{ [0, 0] } union { [1, i]: 1 <= i < N}
```

S, padding, T

- PolyLib: (deals with rational sets, polyhedra)

```
2
2 5
0   1   0   0   0
0   0   1   0   0
3 5
0   1   0   0  -1
1   0   1   0  -1
1   0  -1   1  -1
```

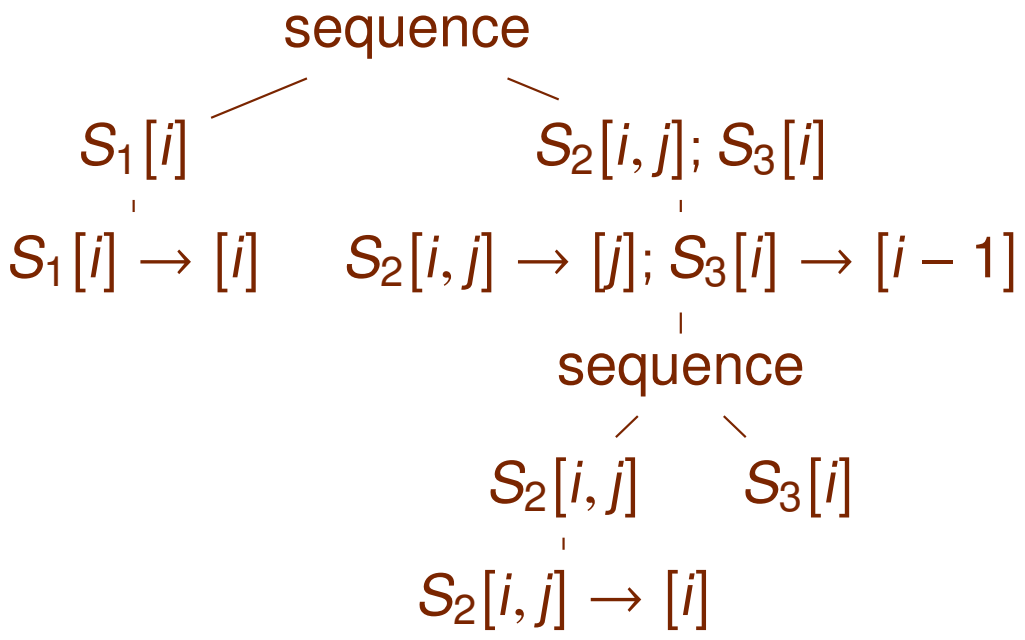
equality/inequality, N

Schedule Representation

[7, 9, 12, 15]

$$
\begin{aligned}
T_1 &: \{[i] \to [0, i]\} \\
T_2 &: \{[i, j] \to [1, j, 0, i]\} \\
T_3 &: \{[i] \to [1, i-1, 1]\}
\end{aligned}
$$

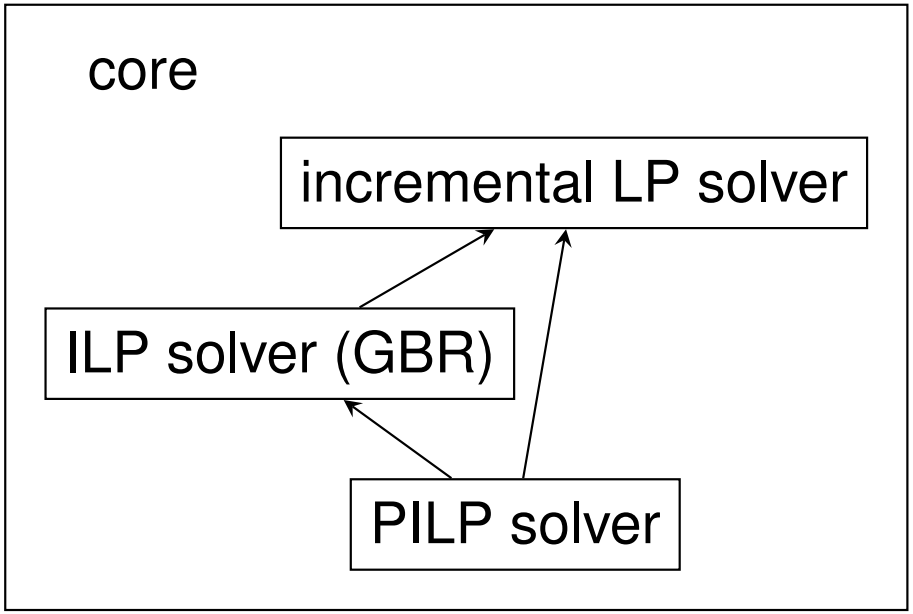
$$
\begin{aligned}
\{\ & S_1[i] \to [0, i, 0, 0]; \\
& S_2[i, j] \to [1, j, 0, i]; \\
& S_3[i] \to [1, i-1, 1, 0]\ \}
\end{aligned}
$$



- Kelly's abstraction
  - schedule spread over statements
- union maps
  - single object
  - schedule transformations can be composed
  - flat schedule space (padding)
- schedule trees
  - single object
  - structured schedule space

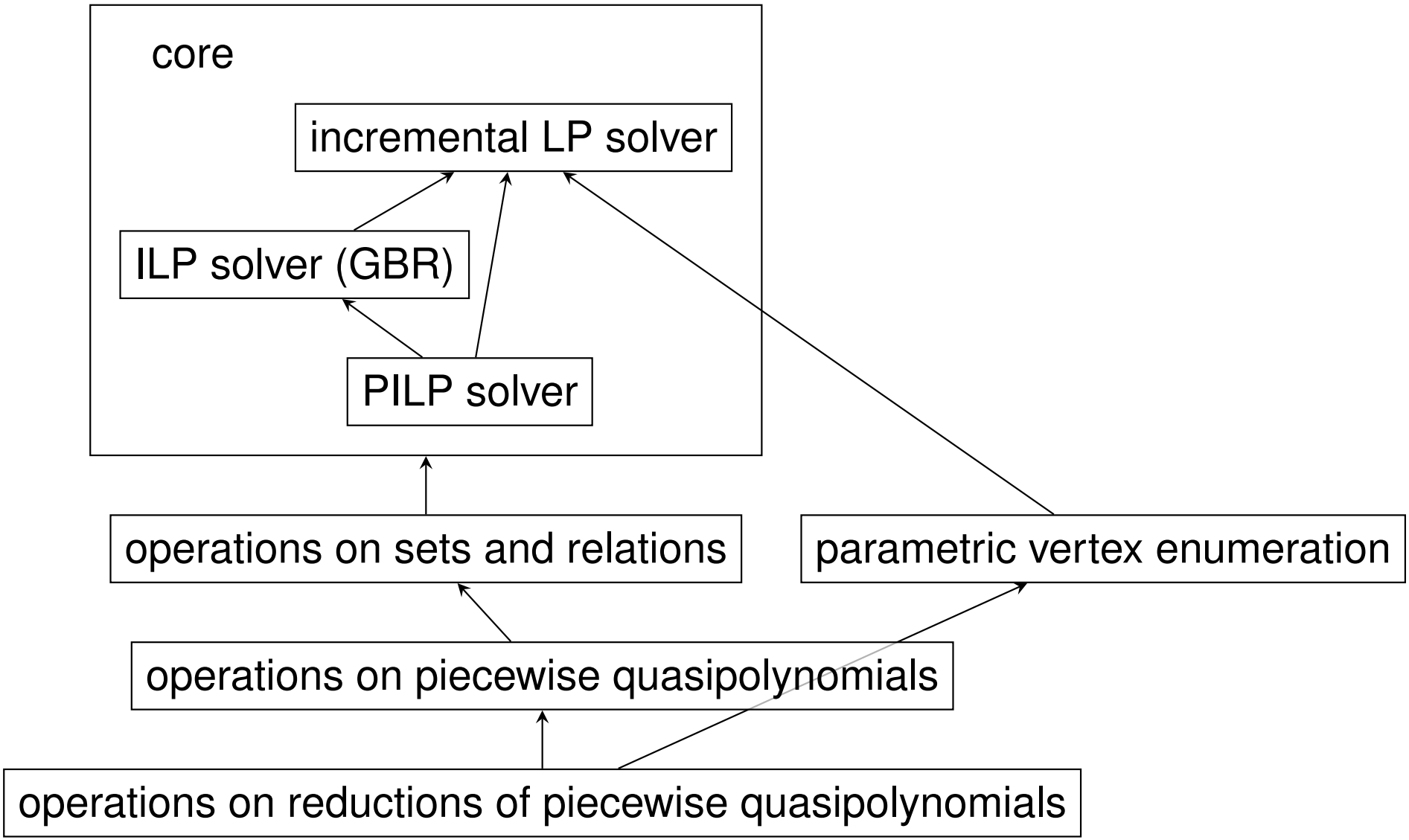
[2, 3, 4, 10, 11, 16]

Internal Structure of isl



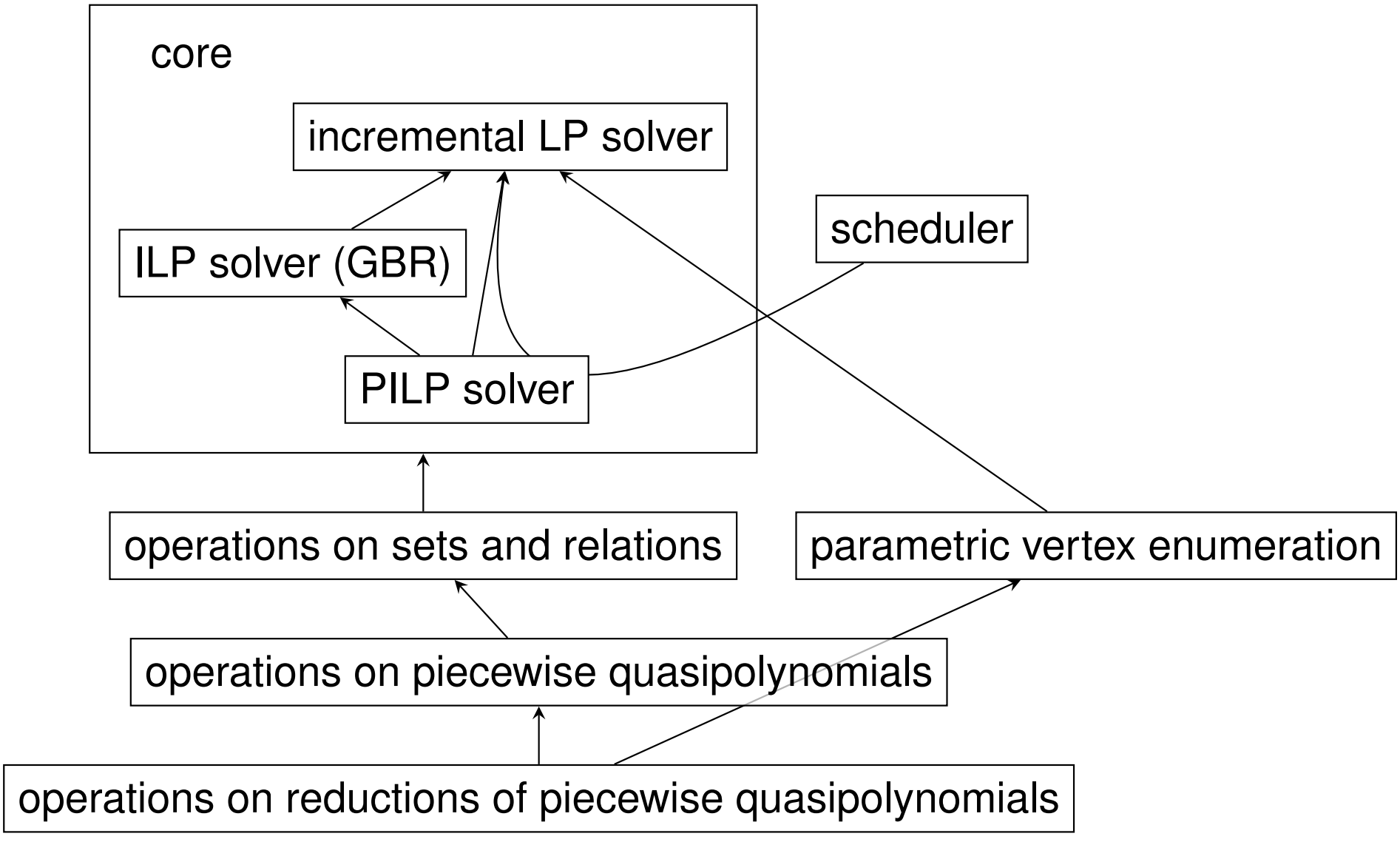
[2, 3, 4, 10, 11, 16]

Internal Structure of isl



[2, 3, 4, 10, 11, 16]

Internal Structure of isl



Internal Representation of Sets and Relations

Each set or relation is stored in disjunctive normal form (DNF)

$$R = \bigcup_i R_i$$

$$R_i = \{ S(\mathbf{i}) \to T(\mathbf{j}) : \exists \mathbf{k} : A_0\mathbf{c} + A_1\mathbf{i} + A_2\mathbf{j} + A_3\mathbf{k} \geq \mathbf{a} \}$$

Each disjunct consists of

- affine equalities and inequalities
- symbolic constants **c**
- local variables **k**
  - existentially quantified, or,
  - integer division $k_i = \lfloor e_i / d_i \rfloor$

Internal Representation of Sets and Relations

Each set or relation is stored in disjunctive normal form (DNF)

$$R = \bigcup_i R_i$$

$$R_i = \{ S(\mathbf{i}) \to T(\mathbf{j}) : \exists \mathbf{k} : A_0\mathbf{c} + A_1\mathbf{i} + A_2\mathbf{j} + A_3\mathbf{k} \geq \mathbf{a} \}$$

Each disjunct consists of

- affine equalities and inequalities
- symbolic constants **c**
- local variables **k**
  - existentially quantified, or,
  - integer division $k_i = \lfloor e_i / d_i \rfloor$

Conversion to DNF

$$\neg(\exists \mathbf{a} : f(\mathbf{x}, \mathbf{a})) \quad \to \quad \neg f(\mathbf{x}, g(\mathbf{x}))$$

$\Rightarrow$ determine unique value of **a** satisfying $f(\mathbf{x}, \mathbf{a})$ and write it as an explicit piecewise quasi affine expression $g(\mathbf{x})$ of **x**

$\Rightarrow$ using parametric integer linear programming

Outline

Scheduling Constraints

[13]

- Validity $\mathbf{a} \rightarrow \mathbf{b}$
  - $\Rightarrow$ statement instance **b** needs to be executed after **a**
  - $\Rightarrow$ $f(\mathbf{b}) \geq f(\mathbf{a})$
- Proximity $\mathbf{a} \rightarrow \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed close to **a**
  - $\Rightarrow$ $f(\mathbf{b}) - f(\mathbf{a})$ as small as possible
- Coincidence $\mathbf{a} \rightarrow \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed together with **a**
  - $\Rightarrow$ $f(\mathbf{b}) = f(\mathbf{a})$
  - $\Rightarrow$ band member only considered “coincident” if it coschedules all pairs

Scheduling Constraints

[13]

- Validity $\mathbf{a} \rightarrow \mathbf{b}$
  - $\Rightarrow$ statement instance **b** needs to be executed after **a**
  - $\Rightarrow$ $f(\mathbf{b}) \geq f(\mathbf{a})$
- Proximity $\mathbf{a} \rightarrow \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed close to **a**
  - $\Rightarrow$ $f(\mathbf{b}) - f(\mathbf{a})$ as small as possible
- Coincidence $\mathbf{a} \rightarrow \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed together with **a**
  - $\Rightarrow$ $f(\mathbf{b}) = f(\mathbf{a})$
  - $\Rightarrow$ band member only considered “coincident” if it coschedules all pairs

Schedule constraints only relevant if coscheduled by outer nodes
Other schedule constraints are said to be *carried* by some outer node

Scheduling Constraints

[13]

- Validity $\mathbf{a} \to \mathbf{b}$
  - $\Rightarrow$ statement instance **b** needs to be executed after **a**
  - $\Rightarrow$ $f(\mathbf{b}) \geq f(\mathbf{a})$
- Proximity $\mathbf{a} \to \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed close to **a**
  - $\Rightarrow$ $f(\mathbf{b}) - f(\mathbf{a})$ as small as possible
- Coincidence $\mathbf{a} \to \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed together with **a**
  - $\Rightarrow$ $f(\mathbf{b}) = f(\mathbf{a})$
  - $\Rightarrow$ band member only considered "coincident" if it coschedules all pairs
- Conditional validity (live-range reordering)
  - condition $\mathbf{b} \to \mathbf{c}$ ($\leftsquigarrow$ flow dependences)
  - conditioned validity $\mathbf{a} \to \mathbf{b}$, $\mathbf{c} \to \mathbf{d}$ ($\leftsquigarrow$ order dependences)

Schedule constraints only relevant if coscheduled by outer nodes
Other schedule constraints are said to be *carried* by some outer node

Dependences and Schedule Constraints

[13]

Traditional dependences

- flow dependences
  - ⇒ validity constraints
  - ⇒ proximity constraints
  - ⇒ coincidence constraints (when parallelism is important)
- false dependences
  - ⇒ validity constraints
  - ⇒ coincidence constraints (when parallelism is important)
  - ⇒ proximity constraints (optional for memory reuse)
- pairs of reads with shared write (“input dependences”)
  - ⇒ proximity constraints (optional)

Live-range reordering

- somewhat different classification of dependences
- slightly different mapping to schedule constraints

For example, PPCG currently

- adds false dependences to proximity constraints for historical reasons
- does not consider input dependences
- uses live-range reordering by default

Schedule Output

[15]

Scheduler produces a schedule tree

Main node types

- *band*: instances are executed according to associated multi-dimensional piecewise quasi-affine partial schedule

  the elements of a band are called its *members*

- *sequence*: children are executed in order

Schedule Output

[15]

Scheduler produces a schedule tree

Main node types

- *band*: instances are executed according to associated multi-dimensional piecewise quasi-affine partial schedule

  the elements of a band are called its *members*

  some of the members are marked *coincident*
- *sequence*: children are executed in order

Schedule Output

[15]

Scheduler produces a schedule tree

Main node types

- *band*: instances are executed according to associated multi-dimensional piecewise quasi-affine partial schedule

  the elements of a band are called its *members*

  some of the members are marked *coincident*
- *sequence*: children are executed in order
- *set*: children may be executed in any order

Scheduling Algorithms

[1, 5]

Scheduling algorithms available in `isl`:

- Feautrier
  - carry as many (groups of) validity constraints as possible
  - fine-grained parallelism
- Pluto-algorithm
  - tilability (through permutability)
  - locality with parallelism as extreme case

Scheduling Algorithms

[1, 5]

Scheduling algorithms available in `isl`:

- Feautrier
  - carry as many (groups of) validity constraints as possible
  - fine-grained parallelism
- Pluto-algorithm
  - tilability (through permutability)
  - locality with parallelism as extreme case

Single step of Feautrier used in case of Pluto-scheduler failure

- forced outer coincidence in band
- bounds on schedule coefficients
- proximity schedule constraints

Scheduling Algorithms

[1, 5]

Scheduling algorithms available in `isl`:

- Feautrier
  - carry as many (groups of) validity constraints as possible
  - fine-grained parallelism
- Pluto-algorithm
  - tilability (through permutability)
  - locality with parallelism as extreme case

Single step of Feautrier used in case of Pluto-scheduler failure

- forced outer coincidence in band
- bounds on schedule coefficients
- proximity schedule constraints

Constraints on Schedule Coefficients

Affine schedule row:

$$f_i(\mathbf{x}) = \boxed{\mathbf{c}_i^{\mathrm{x}}}\boxed{\mathbf{x}} + \boxed{\mathbf{c}_i^{\mathrm{n}}}\boxed{\mathbf{n}} + \boxed{c_i^{\mathrm{c}}}$$

Constraints:

- Validity $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$: $f_j(\mathbf{y}) - f_i(\mathbf{x}) \geq 0$
  Farkas $\to$ constraints on $\mathbf{c}_i^{\mathrm{x}}$, $\mathbf{c}_i^{\mathrm{n}}$ and c_i^{c}

Feautrier Scheduler

[5, 16]

- Carry as many groups of validity constraints $S_i[\mathbf{x}] \rightarrow S_j[\mathbf{y}]$ as possible

$$f_j(\mathbf{y}) - f_i(\mathbf{x}) \geq e_k \qquad \text{with } 0 \leq e_k \leq 1$$

Group k carried if $e_k = 1 \Rightarrow$ minimize $\sum_i (1 - e_k)$

Feautrier Scheduler

[5, 16]

- Carry as many groups of validity constraints $S_i[\mathbf{x}] \rightarrow S_j[\mathbf{y}]$ as possible

$$f_j(\mathbf{y}) - f_i(\mathbf{x}) \geq e_k \qquad \text{with } 0 \leq e_k \leq 1$$

  Group k carried if $e_k = 1 \Rightarrow$ minimize $\sum_i (1 - e_k)$
- Small coefficients $\Rightarrow$ minimize $\sum_{ij} |c^{\mathrm{x}}_{i,j}|$ and $\sum_{ij} c^{\mathrm{n}}_{i,j}$

  Note $|c^{\mathrm{x}}_{i,j}|$ requires extra variables
  - $|c^{\mathrm{x}}_{i,j}| = b_{i,j}$ with $c^{\mathrm{x}}_{i,j} \leq b_{i,j} \wedge -c^{\mathrm{x}}_{i,j} \leq b_{i,j}$, or
  - $|c^{\mathrm{x}}_{i,j}| = c^{\mathrm{x},+}_{i,j} + c^{\mathrm{x},-}_{i,j}$ with $c^{\mathrm{x}}_{i,j} = c^{\mathrm{x},+}_{i,j} - c^{\mathrm{x},-}_{i,j} \wedge c^{\mathrm{x},+}_{i,j}, c^{\mathrm{x},-}_{i,j} \geq 0$

Feautrier Scheduler

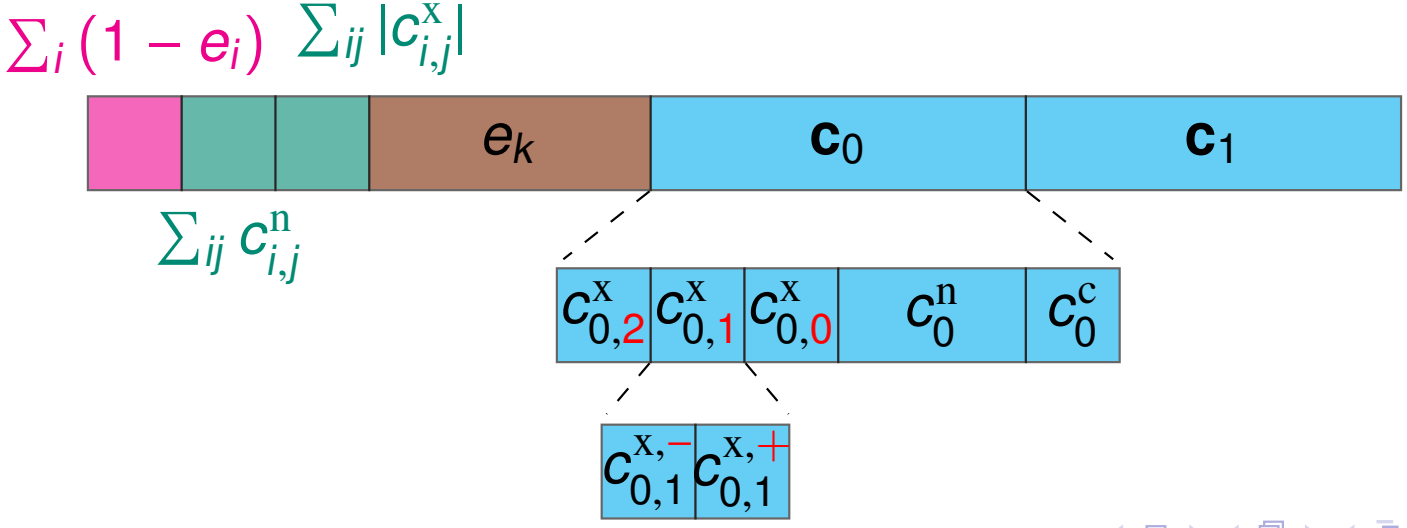
[5, 16]

- Carry as many groups of validity constraints $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$ as possible

$$f_j(\mathbf{y}) - f_i(\mathbf{x}) \geq e_k \qquad \text{with } 0 \leq e_k \leq 1$$

  Group k carried if $e_k = 1 \Rightarrow$ minimize $\sum_i (1 - e_k)$
- Small coefficients $\Rightarrow$ minimize $\sum_{ij} |c^{\mathrm{x}}_{i,j}|$ and $\sum_{ij} c^{\mathrm{n}}_{i,j}$

  Note $|c^{\mathrm{x}}_{i,j}|$ requires extra variables
  - $|c^{\mathrm{x}}_{i,j}| = b_{i,j}$ with $c^{\mathrm{x}}_{i,j} \leq b_{i,j} \wedge -c^{\mathrm{x}}_{i,j} \leq b_{i,j}$, or
  - $|c^{\mathrm{x}}_{i,j}| = c^{\mathrm{x},+}_{i,j} + c^{\mathrm{x},-}_{i,j}$ with $c^{\mathrm{x}}_{i,j} = c^{\mathrm{x},+}_{i,j} - c^{\mathrm{x},-}_{i,j} \wedge c^{\mathrm{x},+}_{i,j}, c^{\mathrm{x},-}_{i,j} \geq 0$
- Encoding



Scheduling Algorithms

[1, 5]

Scheduling algorithms available in `isl`:

- Feautrier
  - carry as many (groups of) validity constraints as possible
  - fine-grained parallelism
- Pluto-algorithm
  - tilability (through permutability)
  - locality with parallelism as extreme case

Single step of Feautrier used in case of Pluto-scheduler failure

- forced outer coincidence in band
- bounds on schedule coefficients
- proximity schedule constraints

Scheduling Algorithms

[1, 5]

Scheduling algorithms available in `isl`:

- Feautrier
  - carry as many (groups of) validity constraints as possible
  - fine-grained parallelism
- Pluto-algorithm
  - tilability (through permutability)
  - locality with parallelism as extreme case

Single step of Feautrier used in case of Pluto-scheduler failure

- forced outer coincidence in band
- bounds on schedule coefficients
- proximity schedule constraints

Constraints on Schedule Coefficients

Affine schedule row:

$$f_i(\mathbf{x}) = \boxed{\mathbf{c}_i^{\mathrm{x}}}\boxed{\mathbf{x}} + \boxed{\mathbf{c}_i^{\mathrm{n}}}\boxed{\mathbf{n}} + \boxed{c_i^{\mathrm{c}}}$$

Constraints:

- Validity $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$: $f_j(\mathbf{y}) - f_i(\mathbf{x}) \geq 0$
  Farkas $\to$ constraints on $\mathbf{c}_i^{\mathrm{x}}$, $\mathbf{c}_i^{\mathrm{n}}$ and c_i^{c}

Constraints on Schedule Coefficients

Affine schedule row:

$$f_i(\mathbf{x}) = \boxed{\mathbf{c}_i^{\mathrm{x}}}\boxed{\mathbf{x}} + \boxed{\mathbf{c}_i^{\mathrm{n}}}\boxed{\mathbf{n}} + \boxed{c_i^{\mathrm{c}}}$$

Constraints:

- Validity $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$: $f_j(\mathbf{y}) - f_i(\mathbf{x}) \geq 0$
  Farkas $\to$ constraints on $\mathbf{c}_i^{\mathrm{x}}$, $\mathbf{c}_i^{\mathrm{n}}$ and c_i^{c}
- Proximity (temporal locality) $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$: $f_j(\mathbf{y}) - f_i(\mathbf{x})$ small
  Farkas $\to$ constraints on $\mathbf{c}_i^{\mathrm{x}}$, $\mathbf{c}_i^{\mathrm{n}}$ and c_i^{c}
- Coincidence (parallelism) $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$: $f_j(\mathbf{y}) - f_i(\mathbf{x}) = 0$
  Farkas $\to$ constraints on $\mathbf{c}_i^{\mathrm{x}}$, $\mathbf{c}_i^{\mathrm{n}}$ and c_i^{c}
- Linear independence of previous rows ($T_{i,0}$): $\mathbf{c}_i^{\mathrm{x}} \neq Y T_{i,0}$

  $\Rightarrow$ compute orthogonal complement of $T_{i,0}$: $U_i T_{i,0}^{\mathrm{t}} = \mathbf{0}$
  $\Rightarrow$ impose $U_i \mathbf{c}_i^{\mathrm{x}} \neq \mathbf{0}$

Pluto Scheduler

[1, 16]

- Uniform bound on proximity constraints $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$:

$$u(\mathbf{n}) = \mathbf{m} \cdot \mathbf{n} + m_0$$

Impose $f_j(\mathbf{y}) - f_i(\mathbf{x}) \leq \mathbf{m} \cdot \mathbf{n} + m_0$ and minimize $\sum_k |m_k|$ and m_0

Pluto Scheduler

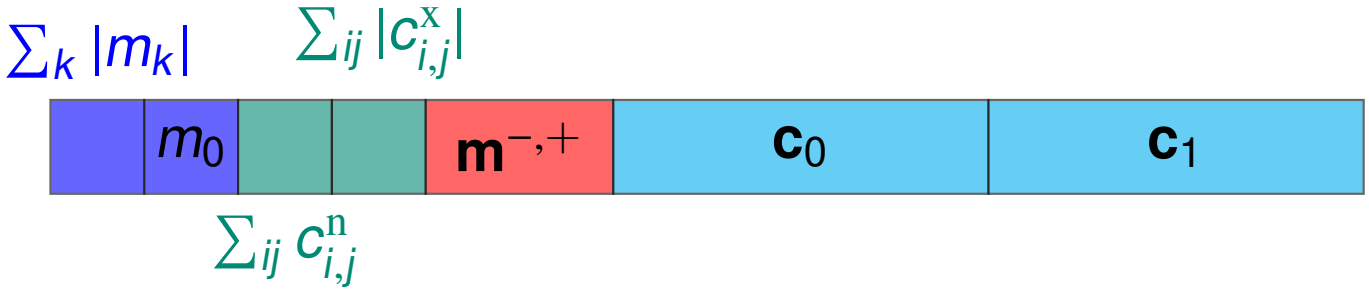
[1, 16]

- Uniform bound on proximity constraints $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$:

$$u(\mathbf{n}) = \mathbf{m} \cdot \mathbf{n} + m_0$$

  Impose $f_j(\mathbf{y}) - f_i(\mathbf{x}) \leq \mathbf{m} \cdot \mathbf{n} + m_0$ and minimize $\sum_k |m_k|$ and m_0

- Encoding



Pluto Scheduler

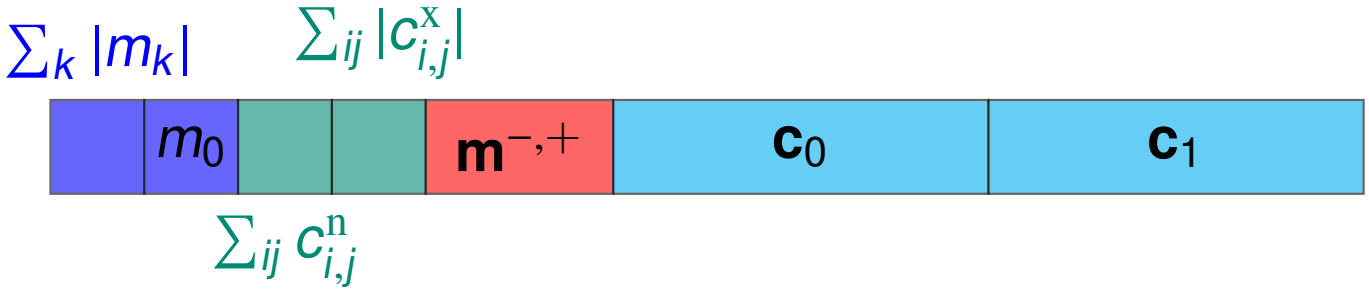
[1, 16]

- Uniform bound on proximity constraints $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$:

$$u(\mathbf{n}) = \mathbf{m} \cdot \mathbf{n} + m_0$$

  Impose $f_j(\mathbf{y}) - f_i(\mathbf{x}) \leq \mathbf{m} \cdot \mathbf{n} + m_0$ and minimize $\sum_k |m_k|$ and m_0

- Encoding



- Linear independence $U\,\mathbf{c}^{\mathrm{x}} \neq \mathbf{0}$
  - $\Rightarrow$ not linear
  - $\Rightarrow$ backtracking search: $U_i\mathbf{c}^{\mathrm{x}} \geq 1$ or $U_i\mathbf{c}^{\mathrm{x}} \leq -1$

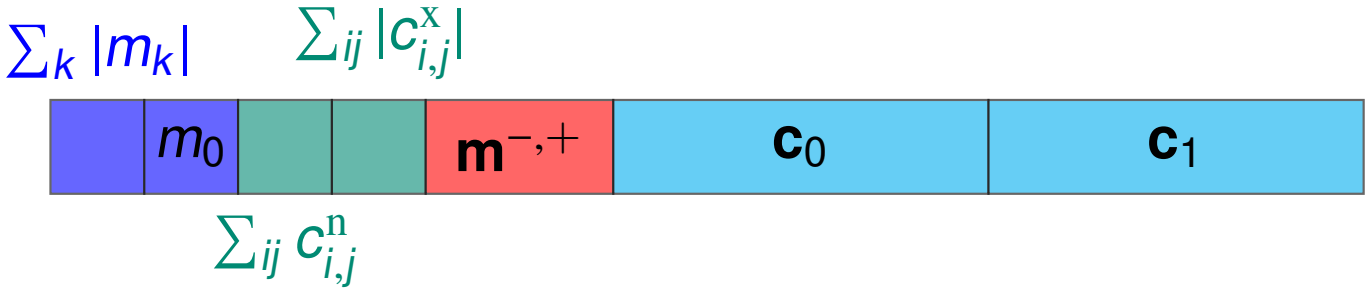
Pluto Scheduler

[1, 16]

- Uniform bound on proximity constraints $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$:

$$u(\mathbf{n}) = \mathbf{m} \cdot \mathbf{n} + m_0$$

  Impose $f_j(\mathbf{y}) - f_i(\mathbf{x}) \leq \mathbf{m} \cdot \mathbf{n} + m_0$ and minimize $\sum_k |m_k|$ and m_0
- Encoding



- Linear independence $U\,\mathbf{c}^{\mathrm{x}} \neq \mathbf{0}$
  - $\Rightarrow$ not linear
  - $\Rightarrow$ backtracking search: $U_i\mathbf{c}^{\mathrm{x}} \geq 1$ or $U_i\mathbf{c}^{\mathrm{x}} \leq -1$
- Coincidence constraints first imposed then relaxed

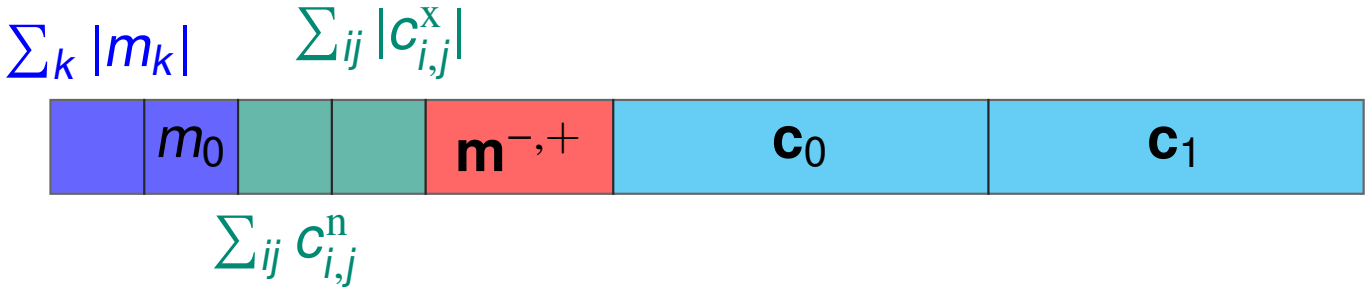
Pluto Scheduler

[1, 16]

- Uniform bound on proximity constraints $S_i[\mathbf{x}] \to S_j[\mathbf{y}]$:

$$u(\mathbf{n}) = \mathbf{m} \cdot \mathbf{n} + m_0$$

  Impose $f_j(\mathbf{y}) - f_i(\mathbf{x}) \leq \mathbf{m} \cdot \mathbf{n} + m_0$ and minimize $\sum_k |m_k|$ and m_0
- Encoding



- Linear independence $U\,\mathbf{c}^{\mathrm{x}} \neq \mathbf{0}$
  - $\Rightarrow$ not linear
  - $\Rightarrow$ backtracking search: $U_i\mathbf{c}^{\mathrm{x}} \geq 1 \quad$ or $\quad U_i\mathbf{c}^{\mathrm{x}} \leq -1$
- Coincidence constraints first imposed then relaxed
- Incremental scheduling
  1. First schedule SCCs separately
  2. Then combine SCCs incrementally

  $\Rightarrow$ better control over coincidence and band depth

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing
  - Schedule may be have large numerators (Feautrier)
  - Schedule may be unnecessarily scaled (Feautrier)
- Proximity constraints may affect feasibility (Pluto)

Known Issues with Scheduling Algorithms

- Scheduling may take a long time

Long Scheduling Times

- Possibly inherent to solution of large (I)LP problems
- Possibly related to Gomory's cutting plane method used in lexicographic dual simplex ILP solver [4]
  On some instances, ILP solver appears to get stuck in seemingly endless cycle

  Alternatives:
  - optimize each optimization criterion c_i in turn, or
  - pick large value M and solve for $\sum_i M^{n-i-1} c_i$

Known Issues with Scheduling Algorithms

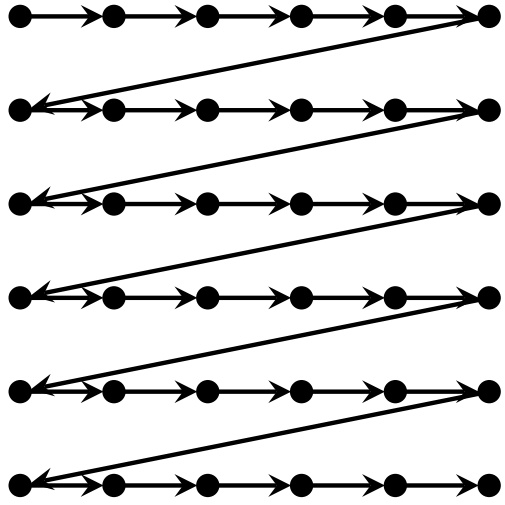
- Scheduling may take a long time

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing

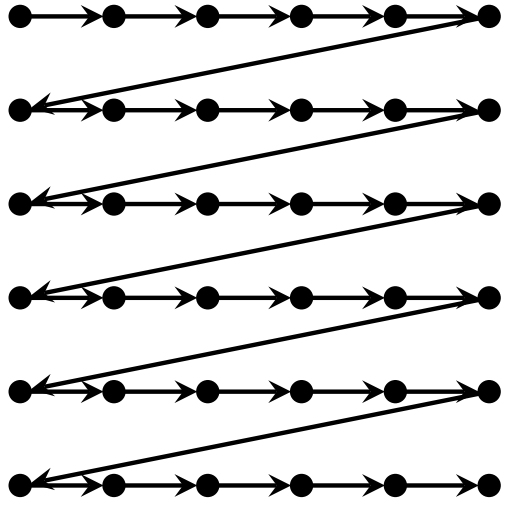
Coalescing

```
for (int i = 0; i < 6; ++i)
  for (int j = 0; j < 6; ++j)
S:  s += f(i, j);
```



Coalescing

```
for (int i = 0; i < 6; ++i)
  for (int j = 0; j < 6; ++j)
S:  s += f(i, j);
```



Valid schedule: $\{ S[i,j] \rightarrow 6i + j \}$

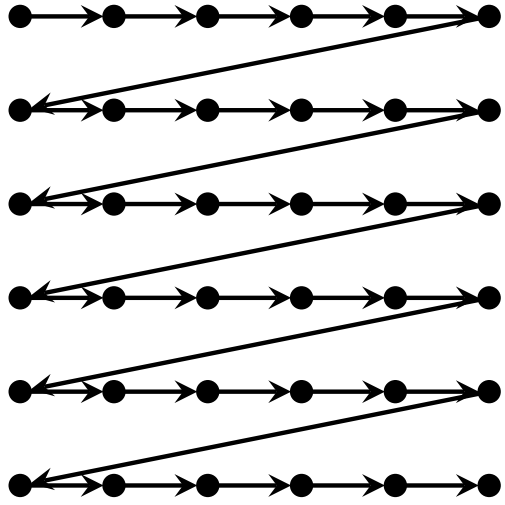
$\Rightarrow$ flattens 2D domain into 1D schedule dimension

$\Rightarrow$ confuses scheduling algorithm

$\Rightarrow$ contains large coefficients

Coalescing

```
for (int i = 0; i < 6; ++i)
  for (int j = 0; j < 6; ++j)
S:  s += f(i, j);
```



Valid schedule: $\{ S[i,j] \rightarrow 6i + j \}$

$\Rightarrow$ flattens 2D domain into 1D schedule dimension

$\Rightarrow$ confuses scheduling algorithm

$\Rightarrow$ contains large coefficients

Handling in `isl`:

- Pluto-algorithm (ILP)
  - $\Rightarrow$ impose bounds on coefficients based on instances set sizes
- Feautrier (LP)
  - $\Rightarrow$ detect coalescing in result and retry with smallest coefficient set to zero

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing
  - Schedule may be have large numerators (Feautrier)

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing
  - Schedule may be have large numerators (Feautrier)

    Optimal solution of rational relaxation may have large numerators
    $\Rightarrow$ continue looking for integer solution

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing
  - Schedule may be have large numerators (Feautrier)

    Optimal solution of rational relaxation may have large numerators
    $\Rightarrow$ continue looking for integer solution
  - Schedule may be unnecessarily scaled (Feautrier)

Scaled Schedules

Feautrier tends to schedule the n statements in an SCC apart

- $\Rightarrow$ $S_i[t, \ldots] \rightarrow n\,t + i$
- $\Rightarrow$ carries maximal number of dependences, but
- $\Rightarrow$ introduces large coefficients

Scaled Schedules

Feautrier tends to schedule the n statements in an SCC apart

- $\Rightarrow$ $S_i[t, \ldots] \rightarrow n\,t + i$
- $\Rightarrow$ carries maximal number of dependences, but
- $\Rightarrow$ introduces large coefficients

In a first attempt, `isl` now only carries self-dependences
$\Rightarrow$ e_k only introduced for validity constraints from node to itself

If result is trivial
$\Rightarrow$ second attempt with all validity constraints

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing
  - Schedule may be have large numerators (Feautrier)

    Optimal solution of rational relaxation may have large numerators
    $\Rightarrow$ continue looking for integer solution
  - Schedule may be unnecessarily scaled (Feautrier)

Known Issues with Scheduling Algorithms

- Scheduling may take a long time
- Schedule may contain large coefficients
  - Schedule may result in loop coalescing
  - Schedule may be have large numerators (Feautrier)

    Optimal solution of rational relaxation may have large numerators
    $\Rightarrow$ continue looking for integer solution
  - Schedule may be unnecessarily scaled (Feautrier)
- Proximity constraints may affect feasibility (Pluto)

Proximity Constraints

Recall:

- Proximity $\mathbf{a} \rightarrow \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed close to **a**
  - $\Rightarrow$ $f(\mathbf{b}) - f(\mathbf{a})$ as small as possible

Proximity Constraints

Recall:

- Proximity $\mathbf{a} \to \mathbf{b}$
  - $\Rightarrow$ statement instance $\mathbf{b}$ preferably executed close to $\mathbf{a}$
  - $\Rightarrow$ $f(\mathbf{b}) - f(\mathbf{a})$ as small as possible

Pluto-algorithm

- looks for uniform bound $f(\mathbf{b}) - f(\mathbf{a}) \leq u(\mathbf{n})$ over all such pairs
- "minimizes" $u(\mathbf{n})$

Proximity Constraints

Recall:

- Proximity $\mathbf{a} \to \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed close to **a**
  - $\Rightarrow$ $f(\mathbf{b}) - f(\mathbf{a})$ as small as possible

Pluto-algorithm

- looks for uniform bound $f(\mathbf{b}) - f(\mathbf{a}) \leq u(\mathbf{n})$ over all such pairs
- "minimizes" $u(\mathbf{n})$

```
A:   a = f1();
     for (int i = 0; i < n; ++i)
B:       A[i] = a;
C:   b = f1();
     for (int i = 0; i < m; ++i)
D:       B[i] = b;
```

Proximity constraints: $\{ \mathtt{A}[] \to \mathtt{B}[i] : 0 \leq i < n; \mathtt{C}[] \to \mathtt{D}[i] : 0 \leq i < m \}$

- $\Rightarrow$ $u(\mathbf{n})$ needs to be larger than n and m
- $\Rightarrow$ $u(\mathbf{n})$ cannot involve m (constraint $\mathtt{A}[] \to \mathtt{B}[i]$ for every value of m)
- $\Rightarrow$ $u(\mathbf{n})$ cannot involve n (constraint $\mathtt{C}[] \to \mathtt{D}[i]$ for every value of n)
- $\Rightarrow$ no non-trivial solution

Proximity Constraints

Recall:

- Proximity $\mathbf{a} \to \mathbf{b}$
  - $\Rightarrow$ statement instance **b** preferably executed close to **a**
  - $\Rightarrow$ $f(\mathbf{b}) - f(\mathbf{a})$ as small as possible

Pluto-algorithm

- looks for uniform bound $f(\mathbf{b}) - f(\mathbf{a}) \leq u(\mathbf{n})$ over all such pairs
- "minimizes" $u(\mathbf{n})$

Note: if some proximity constraint enforces large $u(\mathbf{n})$
then other proximity constraints are essentially ignored

References I

[1] Uday Bondhugula, Muthu Baskaran, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. “Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model”. In: *International Conference on Compiler Construction (ETAPS CC)*. Apr. 2008. DOI: 10.1007/978-3-540-78791-4_9.

[2] William Cook, Thomas Rutherford, Herbert E. Scarf, and David F. Shallcross. *An Implementation of the Generalized Basis Reduction Algorithm for Integer Programming*. Cowles Foundation Discussion Papers 990. available at http://ideas.repec.org/p/cwl/cwldpp/990.html. Cowles Foundation, Yale University, Aug. 1991.

[3] David Detlefs, Greg Nelson, and James B. Saxe. “Simplify: a theorem prover for program checking”. In: *J. ACM* 52.3 (2005), pp. 365–473. DOI: 10.1145/1066100.1066102.

References II

[4] Paul Feautrier. “Parametric Integer Programming”. In: *RAIRO Recherche Opérationnelle* 22.3 (1988), pp. 243–268.

[5] Paul Feautrier. “Some Efficient Solutions to the Affine Scheduling Problem. Part II. Multidimensional Time”. In: *International Journal of Parallel Programming* 21.6 (Dec. 1992), pp. 389–420. DOI: 10.1007/BF01379404.

[6] Tobias Grosser, Armin Größlinger, and Christian Lengauer. “Polly - Performing polyhedral optimizations on a low-level intermediate representation”. In: *Parallel Processing Letters* 22.04 (2012). DOI: 10.1142/S0129626412500107.

[7] Wayne Kelly. *Optimization within a Unified Transformation Framework*. Tech. rep. CS-TR-3725. Dept. of CS, Univ. of Maryland, College Park, 1996.

References III

[8] Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and David Wonnacott. *The Omega Library*. Tech. rep. University of Maryland, Nov. 1996.

[9] Wayne Kelly and William Pugh. "A unifying framework for iteration reordering transformations". In: *IEEE First International Conference on Algorithms and Architectures for Parallel Processing, 1995. ICAPP 95. IEEE First ICA/sup 3/PP.* Vol. 1. 1995, pp. 153–162. DOI: 10.1109/ICAPP.1995.472180.

[10] Vincent Loechner and Doran K. Wilde. "Parameterized Polyhedra and Their Vertices". In: *International Journal of Parallel Programming* 25.6 (Dec. 1997), pp. 525–549. DOI: 10.1023/A:1025117523902.

References IV

[11] Sven V. “isl: An Integer Set Library for the Polyhedral Model”. In: *Mathematical Software - ICMS 2010*. Ed. by Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama. Vol. 6327. Lecture Notes in Computer Science. Springer, 2010, pp. 299–302. DOI: 10.1007/978-3-642-15582-6_49.

[12] Sven V. “Counting Affine Calculator and Applications”. In: *First International Workshop on Polyhedral Compilation Techniques (IMPACT’11)*. Chamonix, France, Apr. 2011. DOI: 10.13140/RG.2.1.2959.5601.

[13] Sven V. and Albert Cohen. “Live-Range Reordering”. In: *Proceedings of the sixth International Workshop on Polyhedral Compilation Techniques*. Prague, Czech Republic, Jan. 2016. DOI: 10.13140/RG.2.1.3272.9680.

References V

[14] Sven V. and Tobias Grosser. “Polyhedral Extraction Tool”. In: *Second International Workshop on Polyhedral Compilation Techniques (IMPACT’12)*. Paris, France, Jan. 2012. DOI: 10.13140/RG.2.1.4213.4562.

[15] Sven V., Serge Guelton, Tobias Grosser, and Albert Cohen. “Schedule Trees”. In: *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*. Vienna, Austria, Jan. 2014. DOI: 10.13140/RG.2.1.4475.6001.

[16] Sven V. and Gerda Janssens. *Scheduling for PPCG*. Report CW 706. Leuven, Belgium: Department of Computer Science, KU Leuven, June 2017. DOI: 10.13140/RG.2.2.28998.68169.

[17] Sven V., Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. “Polyhedral parallel code generation for CUDA”. In: *ACM Trans. Archit. Code Optim.* 9.4 (2013), p. 54. DOI: 10.1145/2400682.2400713.

References VI

[18] Sven V., Rachid Seghir, Kristof Beyls, Vincent Loechner, and Maurice Bruynooghe. “Counting integer points in parametric polytopes using Barvinok’s rational functions”. In: *Algorithmica* 48.1 (June 2007), pp. 37–66. DOI: 10.1007/s00453-006-1231-0.

[19] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven V., Andrew Adams, and Albert Cohen. “Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions”. In: *ArXiv e-prints* (Feb. 2018). arXiv: 1802.04730 [cs.PL].

[20] Doran K. Wilde. *A Library for doing polyhedral operations*. Tech. rep. 785. IRISA, Rennes, France, 1993, 45 p.